DOM

Document Object Model

Le DOM (*Document Object Model*) est une interface de programmation (ou API, *Application Programming Interface*) pour les documents XML et HTML. Via le Javascript, le DOM permet d'accéder au code du document ; on va alors pouvoir modifier des éléments du code HTML.

Un DOM est une description structurée d'un document HTML ou XML.

Un DOM fournit une interface à cette structure et qui permet de modifier structure, contenu et style d'un document XML ou HTML

Un DOM est un arbre avec des noeuds

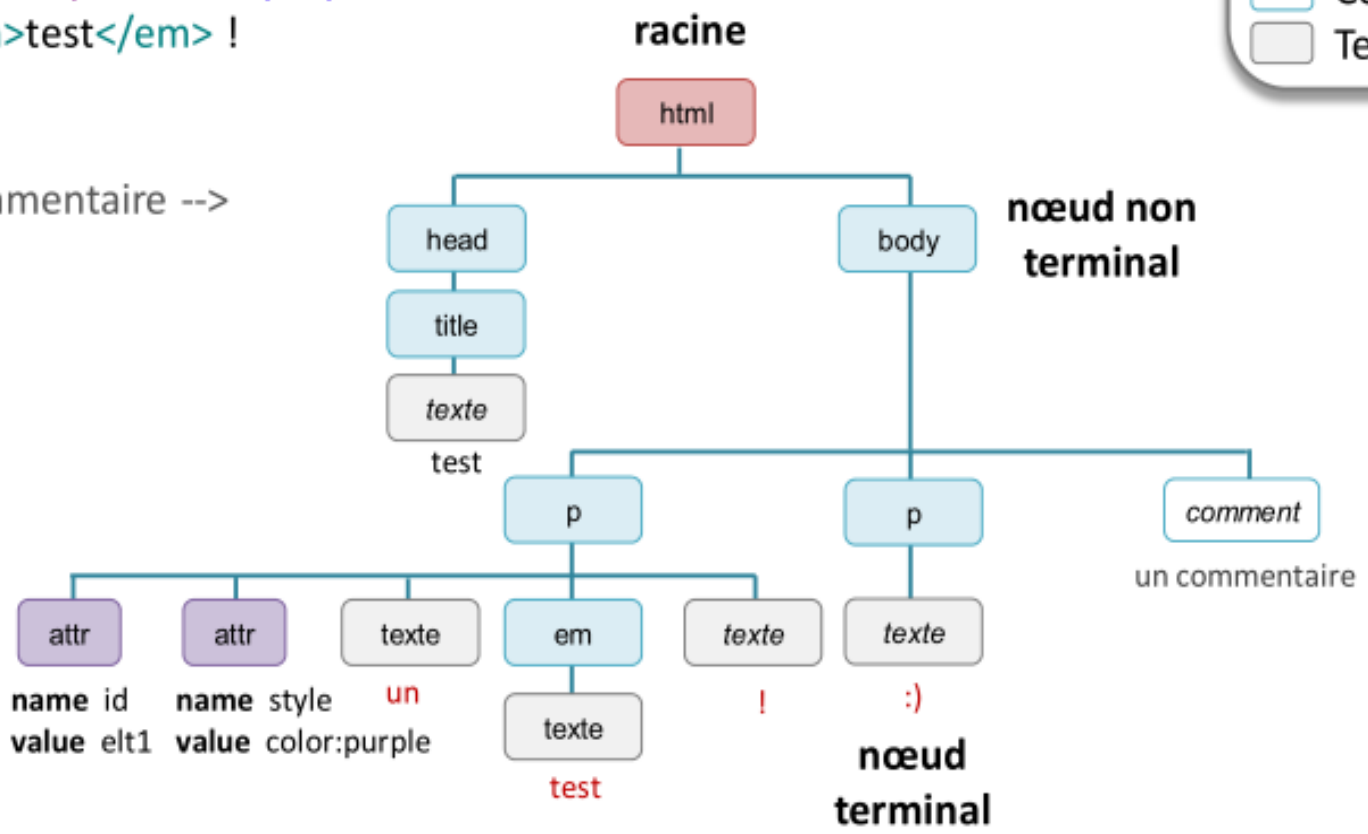
Chaque noed est un objet pour lequel il existe des méthodes et des propriétés (qu'on peut "lire" ou modifier).

DOM = représentation arborescente des documents XML/HTML
+ APIs de navigation et de modification de l'arbre

- Modèle abstrait indépendant du langage

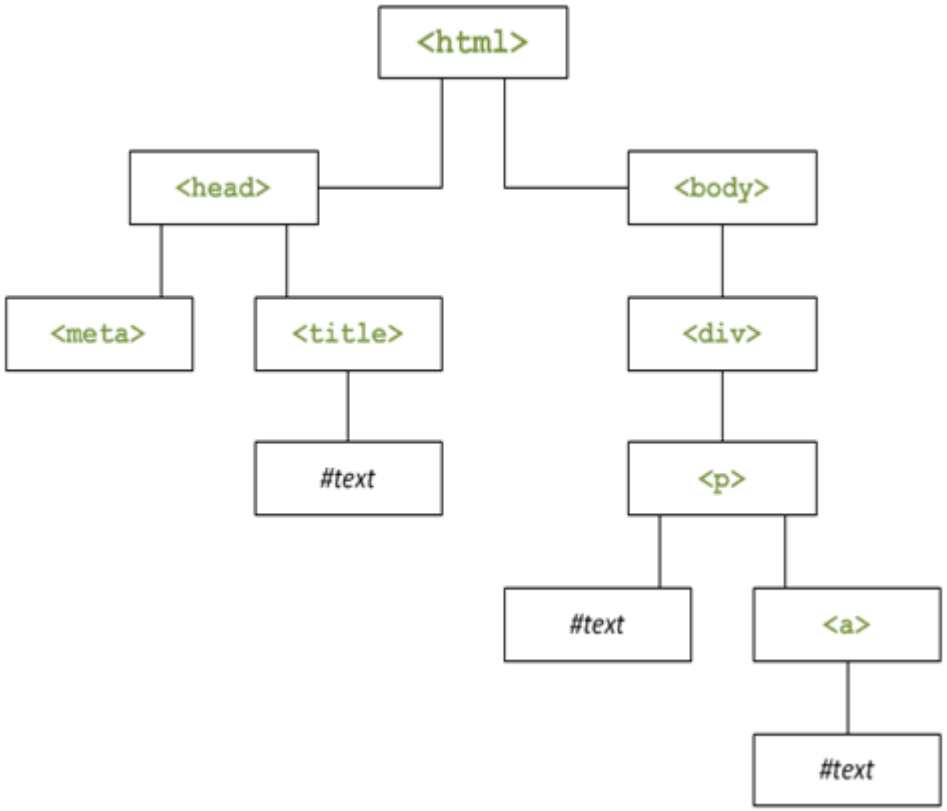
Manipulation des objets du document
DOM – arbre du document

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>test</title>
</head>
<body>
  <p  id="elt1" style="color:purple">
      un <em>test</em> !
  </p>
  <p>:)</p>
  <!-- un commentaire -->
</body>
</html>
```



La structure DOM

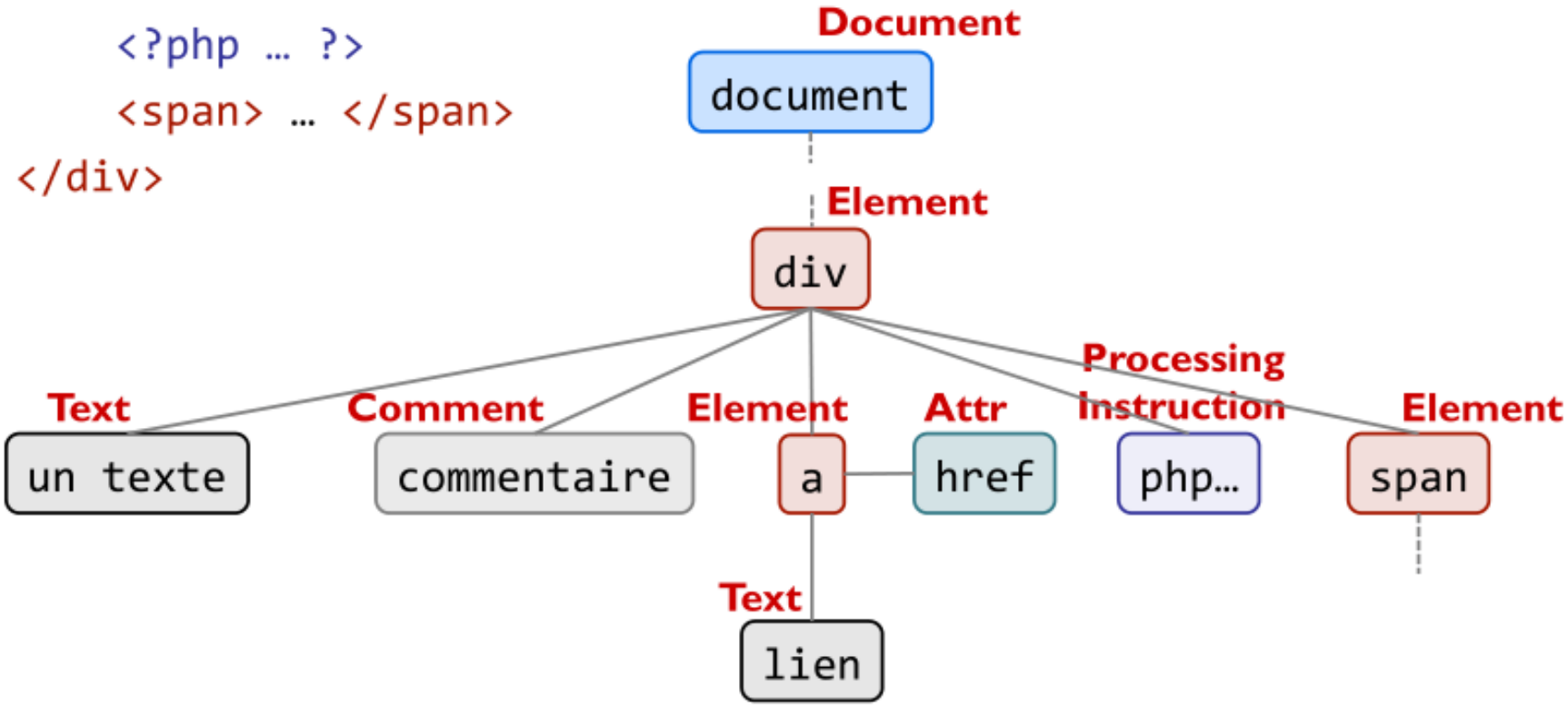
- le DOM pose comme concept que la page Web est vue comme un arbre, comme une hiérarchie d'éléments. On peut donc schématiser une page Web simple comme ceci :




```
1  <!doctype html>
2  <html>
3    <head>
4      <meta charset="utf-8" />
5      <title>Le titre de la page</title>
6    </head>
7
8    <body>
9      <div>
10       <p>Un peu de texte <a>et un lien</a></p>
11     </div>
12   </body>
13 </html>
```

Principaux types de noeuds

```
<div>un texte
    <!-- commentaire -->
    <a href="url">lien</a>
    <?php … ?>
    <span> … </span>
</div>
```



La structure DOM

- L'élément <html> contient deux éléments, appelés **enfants** : <head> et <body>. Pour ces deux enfants, <html> est l'élément **parent**. Chaque élément est appelé **nœud** (*node* en anglais). L'élément <head> contient lui aussi deux enfants : <meta> et <title>. <meta> ne contient pas d'enfant tandis que <title> en contient un, qui s'appelle #text. Comme son nom l'indique, #text est un élément qui contient du texte.
- NB : le texte présent dans une page Web est vu par le DOM comme un nœud de type #text.

Conception de l’arborescence

- Le schéma adopté des documents est une arborescence hiérarchisée.
- Les différentes composantes d'une telle arborescence sont désignées comme étant des nœuds. L'objet central du modèle DOM est pour cette raison l'objet **node (node = nœud).**
- Il existe différents types de nœuds : les nœuds-élément, les nœuds-enfant, nœuds-associé et les nœuds-texte.

Conception de l'arborescence

Exemple HTML :

```
<h1 align="center">Bonjour <i>tout le monde</i></h1>
```

- L'élément h1 est le nœud de départ
- Ce nœud a d'après les règles du modèle DOM, deux nœuds-enfant et un nœud associé :
  - **Les nœuds enfant** sont d'une part *le nœud texte* avec le mot "Bonjour" suivi d'un espace, d'autre part *le nœud élément* de l'élément i.
  - L'attribut align dans la balise ouvrante <h1> n'est pas par contre un nœud-enfant mais un **nœud associé**.

Conception de l’arborescence

Exemple HTML :

```
<h1 align="center">Bonjour <i>tout le monde</i>!</h1>
```

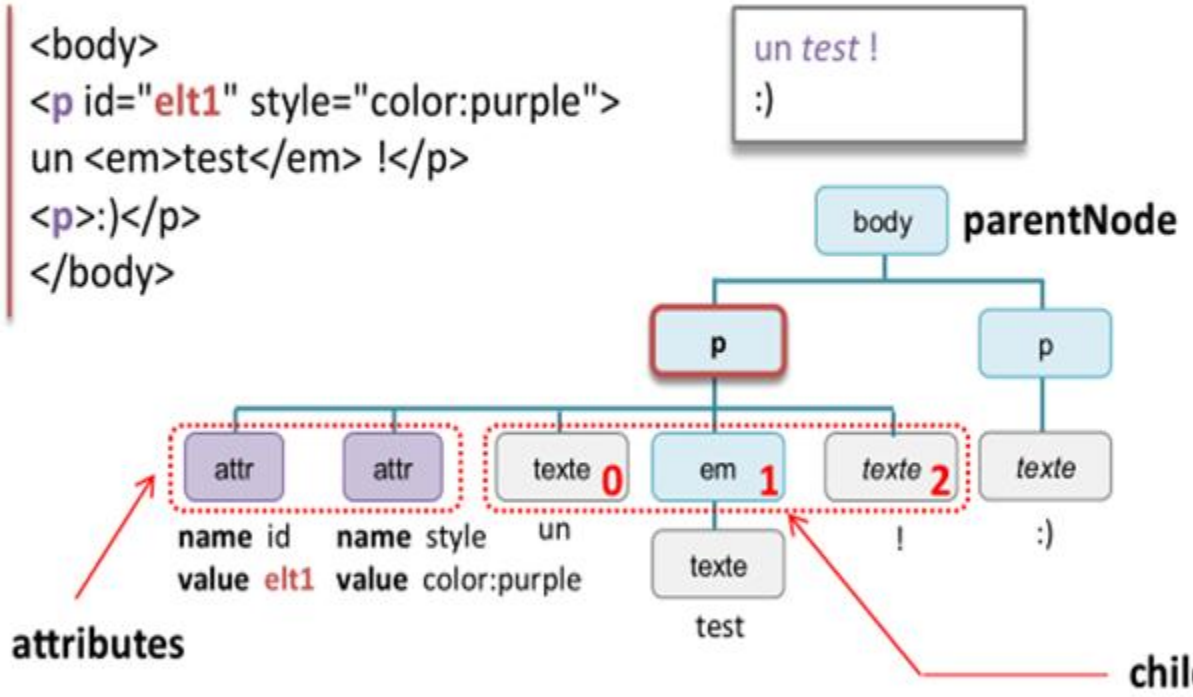
- Le **nœud-attribut** a toutefois lui-même un nœud enfant à savoir la valeur affectée(center).
- Même le nœud-élément de l'élément i a, à son tour, un nœud-enfant, à savoir le **nœud texte** de son contenu de caractères, donc le mot "tout le monde".

accès aux nœuds

Accès par

ID	document.getElementById('*id_element*')	-> noeud
Nom d'élément	document.getElementsByTagName('*nom_element*')	-> collection de nœuds
Valeur de l'attribut name	document.getElementsByName('*val_attribut*')	-> collection de nœuds

```
<body>
<p id="elt1" style="color:purple">
un <em>test</em> !</p>
<p>:)</p>
</body>
```



Collection = tableau de nœuds

prédéfinies : forms, images, links
fils d'un nœud : childNodes
attributs d'un nœud : attributes
nom : getElementsByName
attr. name : getElementsByTagName

accès à un nœud : [i] ou item(i)
avec i >= 0

```
var el = document.getElementById('elt1');
alert(el.tagName);                  // P
alert(el.parentNode.nodeName);      // BODY
alert(el.childNodes[1].nodeName)    // EM
```

```
var tab_el = document.getElementsByTagName('p');
alert(tab_el[0].childNodes[2].nodeName);  // #text
alert(tab_el[0].childNodes[2].nodeValue);    // !
```

accès aux nœuds

En plus des propriétés, les objets du DOM exposent des méthodes qui peuvent être appelés à partir d'un programme en JavaScript (ainsi qu'en d'autres langages de programmation).

Les méthodes et les propriétés définies pour les différents objets du DOM constituent ce qu'on appelle l'interface de programmation du DOM.

Une méthode est, en gros, une opération qui peut être effectuée sur un objet ; une propriété est une variable contenue dans un objet, dont la valeur peut être lue ou modifiée.

Par exemple, la méthode `getElementById()` de l'objet `document` renvoie l'objet qui correspond à l'élément HTML du document ayant l'identifiant (attribut *id* de la balise HTML) souhaité. Ainsi,

```
var element = document.getElementById("NPRIMES");
```

renvoie l'objet correspondant à l'élément du document tel que `id` = NPRIMES.

accès aux nœuds

getElementById() :

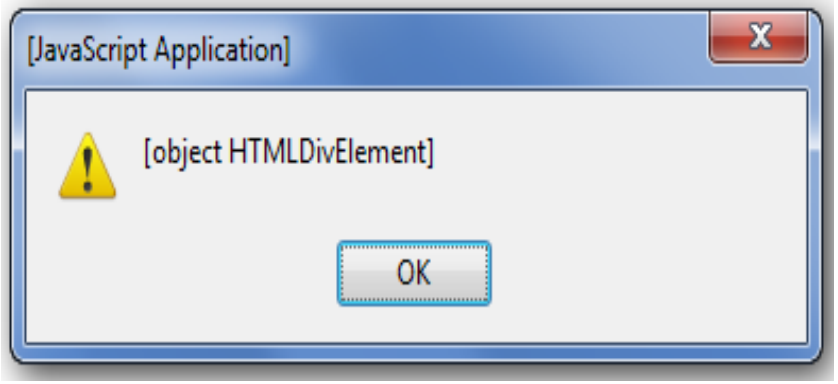
Cette méthode permet de récupérer, s'il existe, le premier élément HTML dont *l'attribut ID* a la valeur spécifiée. Sinon un objet à la valeur Null sera retourné.

Exemple :

```
<div id="myDiv"><p>Un peu de texte <a>et un lien</a></p></div>
<script>
var div = document.getElementById('myDiv');
   alert(div); </script>
```

On nous dit alors que `div` est un objet de type `HTMLDivElement`. Cela fonctionne.

Le résultat est le suivant :



accès aux nœuds

getElementsByTagName() :

- Cette méthode permet de récupérer, sous la forme d'un tableau, *tous les éléments* dont *la balise* a le nom spécifié. Si, dans une page, on veut récupérer tous les , il suffit de faire comme ceci :

```
1 var divs = document.getElementsByTagName('div');
2
3 for (var i = 0, c = divs.length ; i < c ; i++) {
4     alert('Element nº ' + (i + 1) + ' : ' + divs[i]);
5 }
```

- La méthode retourne une collection d'éléments (utilisable de la même manière qu'un tableau). Pour accéder à chaque élément, il est nécessaire de parcourir le tableau avec une petite boucle.

accès aux nœuds

getElementsByName() : Renvoie une liste des éléments ayant le nom donné, elle permet de ne récupérer *que les éléments* qui possèdent un **attribut name** que vous spécifiez.

Exemple:

```
<div name="up">200</div>
<div name="up">145</div>
<div name="other">178</div>
up_divs = document.getElementsByName("up");
// retourne une liste des éléments contenant l'attribut name égal à "up"
```

accès aux nœuds

getElementsByClassName() : Renvoie une liste des éléments ayant le nom de la classe donnée

Exemple:

```
<div class="entete">l’entête de la page</div>
<div class="corps">ceci est le corps du texte</div>
<div name=" entete ">une autre entête</div>
up_divs = document.getElementsByClassName(" entete ");
// retourne une liste des éléments contenant l’attribut class égal à "entete “ (1ere et 3ème division)
```

accès aux nœuds

querySelector() et querySelectorAll():

- simplifier la sélection d'éléments dans l'arbre DOM.
- Ces deux méthodes prennent pour paramètre une chaîne de caractères.
- Cette chaîne de caractères doit être un sélecteur CSS.
- La première, querySelector(), renvoie le premier élément trouvé correspondant au sélecteur CSS, tandis que querySelectorAll() va renvoyer *tous* les éléments (sous forme de tableau) correspondant au sélecteur CSS fourni.

Exemple

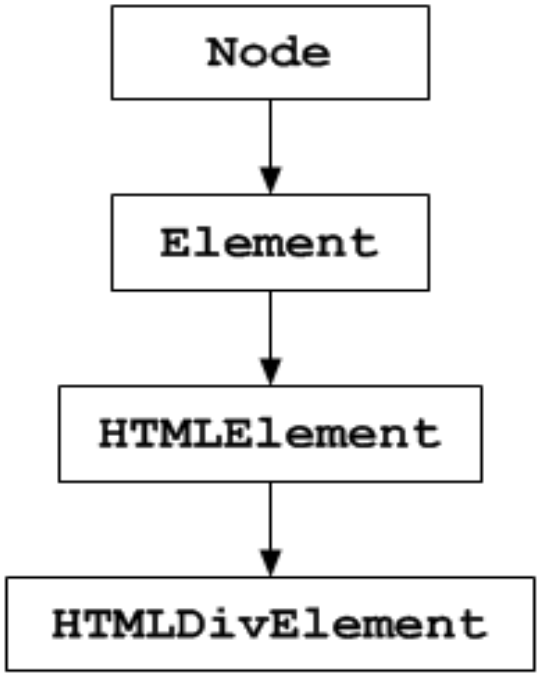
```html
<div id="menu">

  <div class="item">
    <span>Élément 1</span>
    <span>Élément 2</span>
  </div>

  <div class="publicite">
    <span>Élément 3</span>
    <span>Élément 4</span>
  </div>

</div>

<div id="contenu">
  <span>Introduction au contenu de la page...</span>
</div>
```

```javascript
var query = document.querySelector('#menu .item span'),
    queryAll = document.querySelectorAll('#menu .item span');

alert(query.innerHTML); // Affiche : "Élément 1"

alert(queryAll.length); // Affiche : "2"
alert(queryAll[0].innerHTML + ' - ' + queryAll[1].innerHTML); //
```

Résultat : Elément 1- Elément 2

Notion d’héritage

- Un élément est un objet HTMLDivElement, mais un objet, en Javascript, peut appartenir à différents groupes.
- Ainsi, notre est un HTMLDivElement, qui est un sous-objet d'HTMLElement qui est lui-même un sous-objet d'Element. Element est enfin un sous-objet de Node.
- L'objet Node apporte un certain nombre de propriétés et de méthodes qui pourront être utilisées depuis un de ses sous-objets.
- En clair, les sous-objets *héritent* des propriétés et méthodes de leurs objets parents.



Editer les éléments HTML

On suit le schéma suivant :

Node > Element >HTMLElement >HTMLDivElement

On peut jouer sur les attributs d'une balise HTML avec l'objet `Element` et `getAttribute()` et `setAttribute()`, permettant par exemple de modifier un lien :

```
    <a id="myLink" href="http://www.un_lien_quelconque.com">Un lien modifié
dynamiquement</a>
    <script>
       var link = document.getElementById('myLink');
       var href = link.getAttribute('href'); // On récupère l'attribut « href »
       alert(href);
       link.setAttribute('href', 'http://blog.crdp-versailles.fr/rimbaud/'); // on édite
    </script>
```

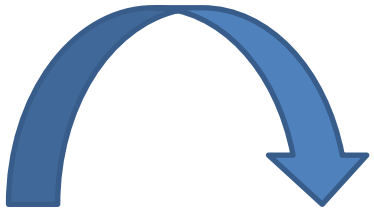
Les attributs : Via l'objet Element

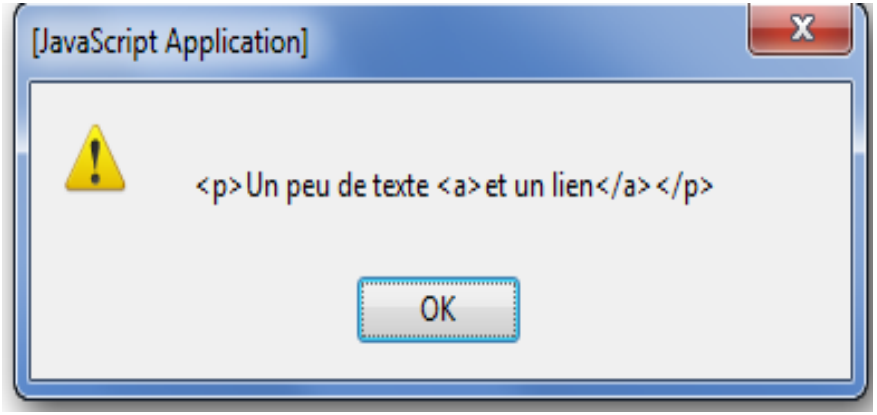
Pour interagir avec les attributs, l'objet Element nous fournit deux méthodes: getAttribute() et setAttribute() permettant respectivement de récupérer et d'éditer un attribut. Le 1er paramètre est le nom de l'attribut, et le 2ème paramètre, dans le cas de setAttribute() , nom de l'attribut avec la nouvelle valeur à donner à l'attribut

Editer les éléments HTML

Récupérer/modifier du HTML : *innerHTML* permet de récupérer/modifier le code HTML enfant d'un élément sous forme de texte. Ainsi, si des balises sont présentes, innerHTML les retournera sous forme de texte :

```
1  <body>
2    <div id="myDiv">
3      <p>Un peu de texte <a>et un lien</a></p>
4    </div>
5
6    <script>
7      var div = document.getElementById('myDiv');
8
9      alert(div.innerHTML);
10   </script>
11 </body>
```





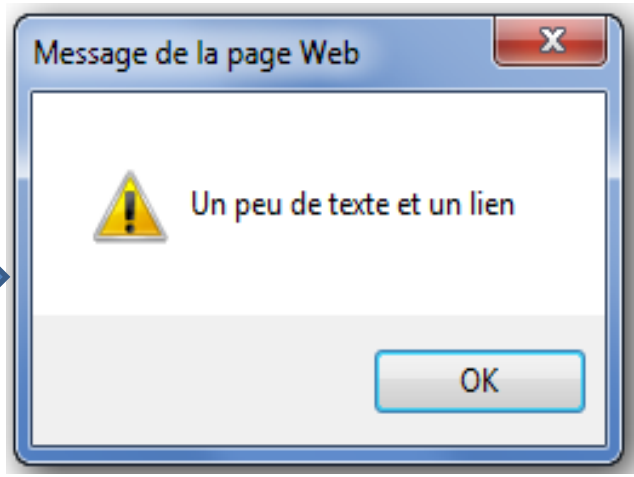
Modifier le contenu HTML

- Pour modifier le contenu HTML, il suffit de faire l'inverse, c'est-à-dire de définir un nouveau contenu :

```
document.getElementById('myDiv').innerHTML = '<blockquote>
Je mets une citation à la place du paragraphe</blockquote>';
```

Récupérer/modifier du texte: <u>textContent</u> récupère le texte (du nœud et ses descendants), pas les balises.

```
1  <body>
2    <div id="myDiv">
3      <p>Un peu de texte <a>et un lien</a></p>
4    </div>
5
6    <script>
7      var div = document.getElementById('myDiv');
8
9      alert(div.textContent);
10   </script>
11 </body>
```



A retenir

- Le DOM va servir à accéder aux éléments HTML présents dans un document afin de les modifier et d'interagir avec eux.
- L'objet window est un objet global qui représente la fenêtre du navigateur
- Document est un sous objet de window et représente la page Web. C'est grâce à lui que l'on va pouvoir accéder aux éléments HTML de la page Web.
- Méthodes,comme getElementById(), getElementsByTagName(),querySelector() ou querySelectorAll(), sont disponibles pour accéder aux éléments.
- Les attributs peuvent tous être modifiés grâce à setAttribute().
- La propriété innerHTML permet de récupérer ou de définir le code HTML présent à l'intérieur d'un élément.
- De son côté, innerText n’est capable que de définir ou récupérer du texte brut, sans aucunes balises HTML.

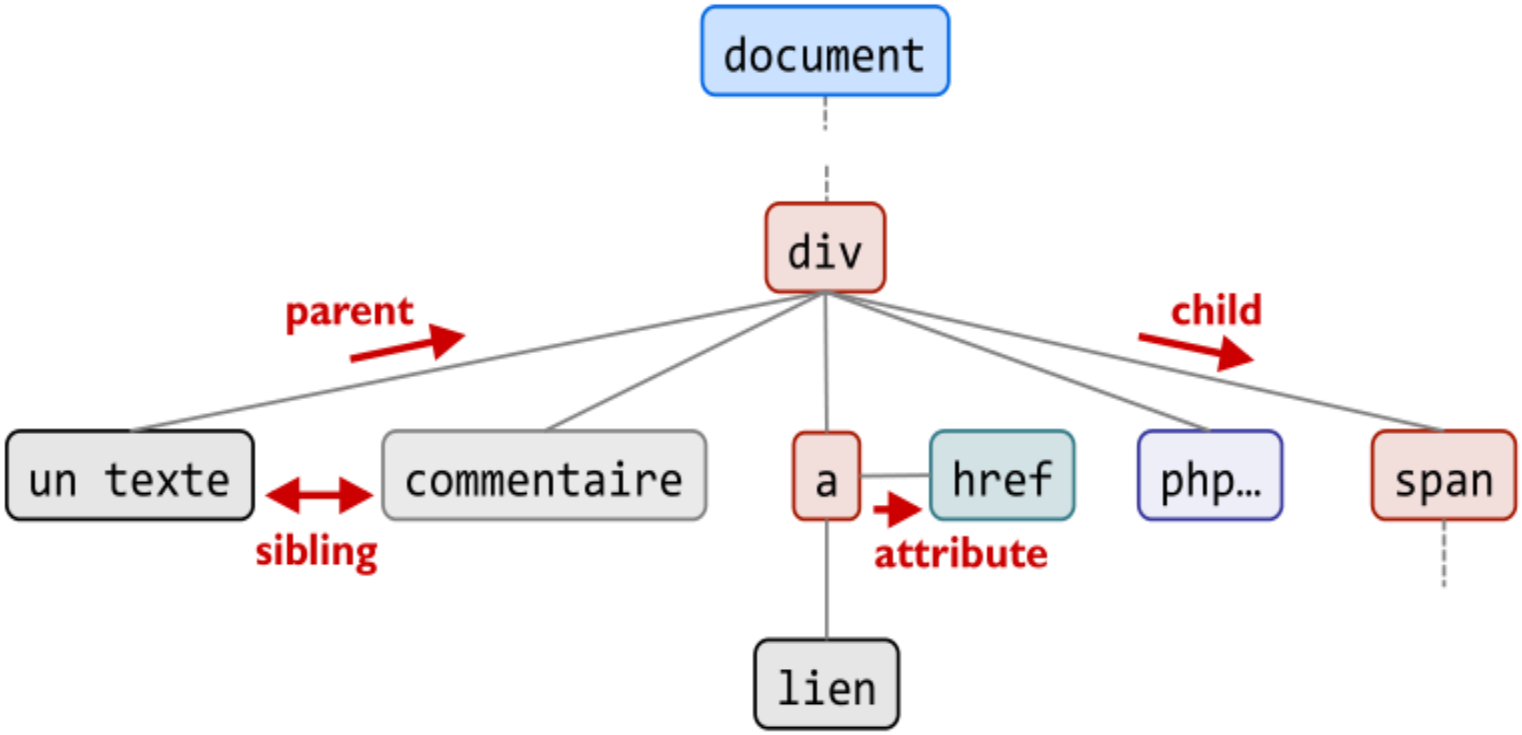
Naviguer entre les nœuds

Selon le standard DOM, tout est un nœud dans un document HTML :
- le document dans son complexe est un nœud document ;
- chaque élément HTML est un nœud élément ;
- le texte dans les éléments HTML est constitué par des nœuds texte ;
- chaque attribut HTML est un nœud attribut ;
- même les commentaires sont des nœuds commentaire.

On peut accéder à tous les nœuds de l'arbre avec JavaScript. Tous les nœuds peuvent être modifiés et il est possible d'éliminer des nœuds ou d'en créer des nouveaux.

Les relations parmi les nœuds de l'arbre sont décrites en termes de parents (*parent* en anglais), fils (*children* en anglais, *child* au singulier) et frères (*siblings* en anglais). La racine de l'arbre est l'unique nœud ne possédant pas de parent.

Naviguer entre les nœuds



A partir de l'objet document

- L'objet document propose des « raccourcis » pour accéder à certains éléments d'une page web
- A partir du nœud document « d » :

```
d.head : élément <head>
d.title : élément <title>

d.body : élément <body>

d.images : collection d'éléments <img>
d.links : collection d'éléments <a>
d.forms : collection d'éléments <form>
```

Des nœuds qui ont le même parent sont des frères. Une chose importante à retenir est que dans l'arbre DOM les fils d'un node sont ordonnés. On peut ainsi parler du premier fils (*first child*) et du dernier fils (*last child*), du frère précédent (*previous sibling*) et du frère suivant (*next sibling*).

Dans la terminologie DOM, on utilise des conventions d'appellation pour se référer aux nœuds qui sont en relation avec un nœud donné :

- `parentNode` est le parent ;
- `childNodes` est la liste des fils ;
- `firstChild` est le premier fils ;
- `lastChild` est le dernier fils ;
- `previousSibling` est le frère précédent ;
- `nextSibling` est le frère suivant ;

Exemple

Considérons le fragment d'HTML suivant :

```
<html>
  <head>
    <title>Introduction à DOM</title>
  </head>
  <body>
    <h1>Première séance</h1>
    <p>Bonjour !</p>
  </body>
</html>
```

Sur la base de ce fragment, on peut affirmer que :

- le nœud `<html>` n'a pas de parent : c'est la racine de l'arbre ;
- le `parentNode` des nœuds `<head>` et `<body>` est le nœud `<html>` ;
- le `parentNode` du nœud texte "Bonjour !" est le nœud `<p>` ;

par ailleurs,

- le nœud `<html>` a deux `childNodes` : `<head>` et `<body>` ;
- le nœud `<head>` a un `childNode` : le nœud `<title>` ;
- le nœud `<title>` aussi n'a qu'un `childNode` : le nœud texte "Introduction à DOM" ;

```
<html>
  <head>
    <title>Introduction à DOM</title>
  </head>
  <body>
    <h1>Première séance</h1>
    <p>Bonjour !</p>
  </body>
</html>
```

- le nœud `<h1>` est `previousSibling` de `<p>` et ce dernier est `nextSibling` de `<h1>` ; les deux sont `childNodes` de `<body>` ;
- l'élément `<head>` est le `firstChild` de l'élément `<html>` ;
- l'élément `<body>` est le `lastChild` de l'élément `<html>` ;
- l'élément `<h1>` est le `firstChild` de l'élément `<body>` ;
- l'élément `<p>` est le `lastChild` de l'élément `<body>`.

La propriété parentNode

La propriété `parentNode` permet d'accéder à l'élément parent d'un élément :

```
<blockquote>
  <p id="myP">Ceci est un paragraphe !</p>
</blockquote>
<script>
  var paragraph = document.getElementById('myP');
  var blockquote = paragraph.parentNode;
</script>
```

Pour accéder à l'élément <blockquote>, qui est le parent de myP. Il suffit d'accéder à myP puis à son parent, avec parentNode.

`nodeType` et `nodeName` permettent de vérifier le type et le nom d'un nœud :

```
var paragraph = document.getElementById('myP');
alert(paragraph.nodeType + '\n\n' +
paragraph.nodeName.toLowerCase());
```

nodeType : retourne les type d'un nœud sous forme d'un nombre entier. Voici un tableau qui liste les types courants, ainsi que leurs numéros.

Numéro	Type
1	nœud élément
2	Nœud attribut
3	Nœud texte
8	Nœud commentaire
9	Nœud document

nodeName : Selon le type de nœud, sa valeur est,

– pour un nœud élément : le nom de sa balise, en toutes majuscules;
– pour un nœud attribut : le nom de l'attribut;
– pour un nœud texte : toujours #text;
– pour le nœud document : toujours #document.

`firstChild` et `lastChild` permettent d'accéder au premier et au dernier élément d'un nœud :

```
<div>
  <p id="myP">Un peu de texte, <a>un lien</a> et <strong>une portion en emphase</strong></p>
</div>
<script>
  var paragraph = document.getElementById('myP');
  var first = paragraph.firstChild;
  var last = paragraph.lastChild;
  alert(first.nodeName.toLowerCase());
  alert(last.nodeName.toLowerCase());
</script>
```

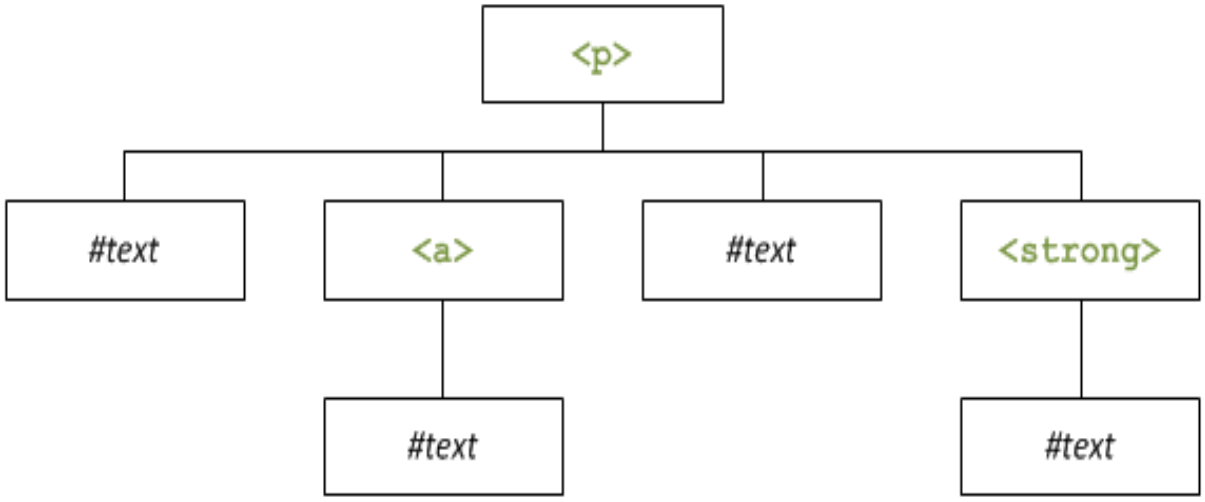


Schéma de l'élément myP

Le premier enfant de <p> est un nœud textuel, alors que le dernier enfant est un élément .

childNodes : retourne un tableau contenant la liste des enfants d'un élément.

nodeValue et **data** : Retourne la valeur d'un nœud.

les propriétés nodeValue et data ne s'appliquent *que* sur des *nœuds textuels*

```
1 var paragraph = document.getElementById('myP');
2 var first = paragraph.firstChild;
3 var last  = paragraph.lastChild;
4
5 alert(first.nodeValue);
6 alert(last.firstChild.data);
```

first contient le 1 nœud, un nœud textuel. Il suffit de lui appliquer la propriété **nodeValue** (ou **data**) pour récupérer son contenu.

Last contient . il faut d'abord accéder au nœud textuel que contient notre élément. Pour cela, on utilise firstChild, ensuite on récupère le contenu avec nodeValue ou data.

Résultat : Un peu de texte, une portion en emphase

`nextSibling` et `previousSibling` permettent d'accéder à l'élément suivant, et au précédent :

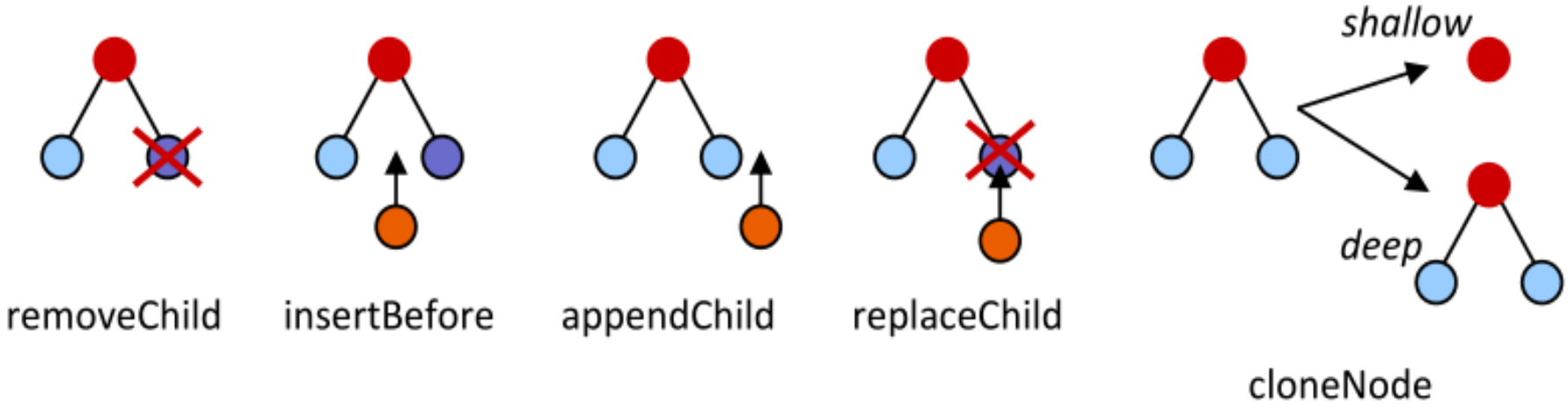
```
<div>
  <p id="myP">Un peu de texte, <a>un lien</a> et <strong>une portion en emphase</strong></p>
</div>
<script>
  var paragraph = document.getElementById('myP');
  var first = paragraph.firstChild;
  var next = first.nextSibling;
  alert(next.firstChild.data); // Affiche « un lien »
</script>
```

Remarque :Les espaces et retours à la ligne effectués dans le code HTML sont généralement considérés comme des nœuds textuels par les navigateurs.

Modification de l'arbre DOM

- Sur n'importe quel nœud « n » :

```
n.removeChild(nodeToRemove) → objet supprimé
n.insertBefore(nodeToInsert, childRef) → objet inséré
n.appendChild(nodeToAppend) → objet inséré
n.replaceChild(newNode, oldNode) → objet remplacé
n.cloneNode(deepOrNot) → objet résultat de la copie
```



Créer et insérer des éléments

Avec le DOM, l'ajout d'un élément HTML se fait en trois temps :

- On crée l'élément.
- On lui affecte des attributs.
- On l'insère dans le document

1. Création de l'élément

La création d'un élément se fait avec la méthode *createElement(),* un sous-objet de l'objet racine, c'est-à-dire document dans la majorité des cas :

```
1 var newLink = document.createElement('a');
```

Créer et insérer des éléments

2. Affectation des attributs :

on définit les attributs, soit avec *setAttribute()*, soit directement avec les propriétés adéquates.

```
newLink.id    = 'mns_link';
newLink.href  = 'http://www.monsite.com';
newLink.title = 'Découvrez mon site !';
```

Créer et insérer des éléments

3. Insertion de l'élément :

On utilise la méthode *appendChild()* pour insérer l'élément.

appendchild signifie « ajouter un enfant », ce qui signifie qu'il nous faut connaître l'élément auquel on va ajouter l'élément créé. Considérons donc le code suivant :

```
1  <!doctype html>
2  <html>
3    <head>
4      <meta charset="utf-8" />
5      <title>Le titre de la page</title>
6    </head>
7
8    <body>
9      <div>
10       <p id="myP">Un peu de texte <a>et un lien</a></p>
11     </div>
12   </body>
13 </html>
```

Créer et insérer des éléments

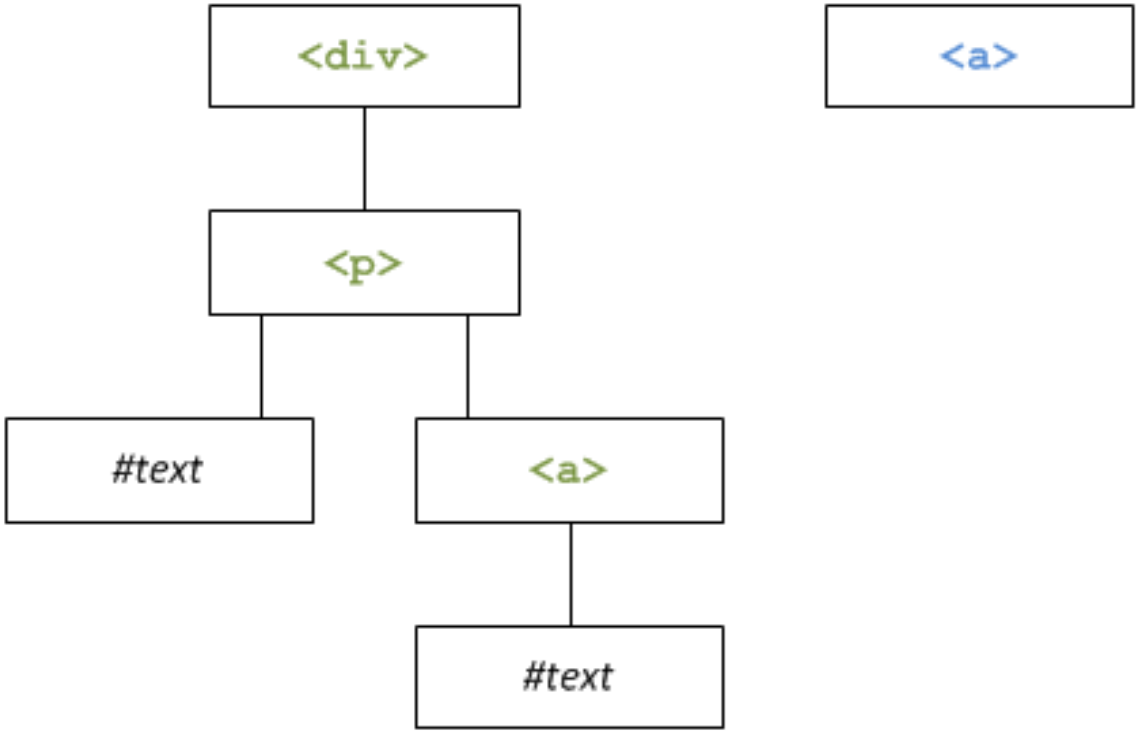
Insertion de l'élément :

- On va ajouter notre élément <a> dans l'élément <p> portant l'ID myP.
- il suffit de récupérer cet élément (p), et d'ajouter notre élément <a> via appendChild() :

```
1 document.getElementById('myP').appendChild(newLink);
```

Créer et insérer des éléments

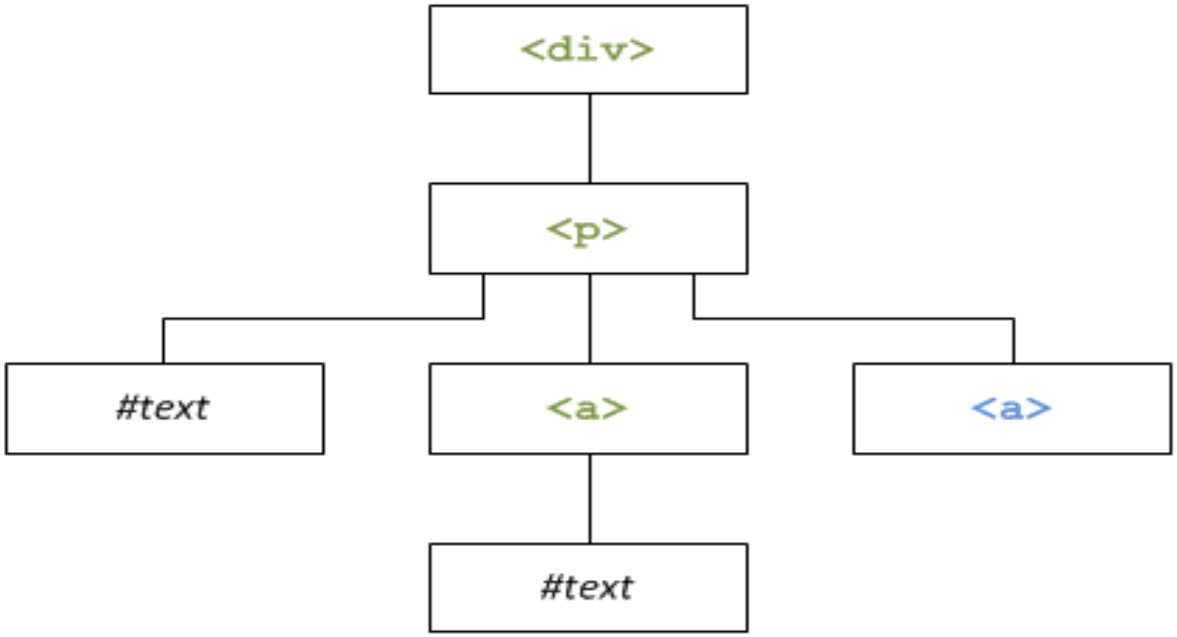
Explication : avant d'insérer notre élément <a>, la structure DOM du document ressemble à ceci :



Avant d'insérer notre élément <a>, la structure DOM du document ressemble à cette figure

Créer et insérer des éléments

On voit que l'élément <a> existe, mais n'est pas lié. Un peu comme s'il était libre dans le document : il n'est pas encore placé. Le but est de le placer comme enfant de l'élément myP. La méthode appendChild() va alors déplacer notre <a> pour le placer en tant que dernier enfant de myP :



L'élément est placé en tant que dernier enfant

Créer et insérer des éléments

Ajouter des nœuds textuels :

L'élément a été inséré, seulement il manque le contenu textuel. La méthode *createTextNode()* sert à créer un nœud textuel (de type #text) qu'il nous suffira d'ajouter à notre élément inséré, comme ceci :

```
var newLinkText = document.createTextNode("mon site");

newLink.appendChild(newLinkText);
```

On l'insère dans le document :

```
<div><p id="myP">Un peu de texte <a>et un lien</a></p></div>
<script>
   var newLink = document.createElement('a');
   newLink.id = 'sdz_link';
   newLink.href = 'http://blog.crdp-versailles.fr/rimbaud/';
   newLink.title = 'Découvrez le blog de la Classe Actu !';
  document.getElementById('myP').appendChild(newLink); // le nouvel élément
est le dernier enfant dans le paragraphe avec id 'myP'
   var newLinkText = document.createTextNode("Le Tonnerre de Rimbaud");
   newLink.appendChild(newLinkText); // ces deux lignes pour ajouter le texte
</script>
```

Résultat :

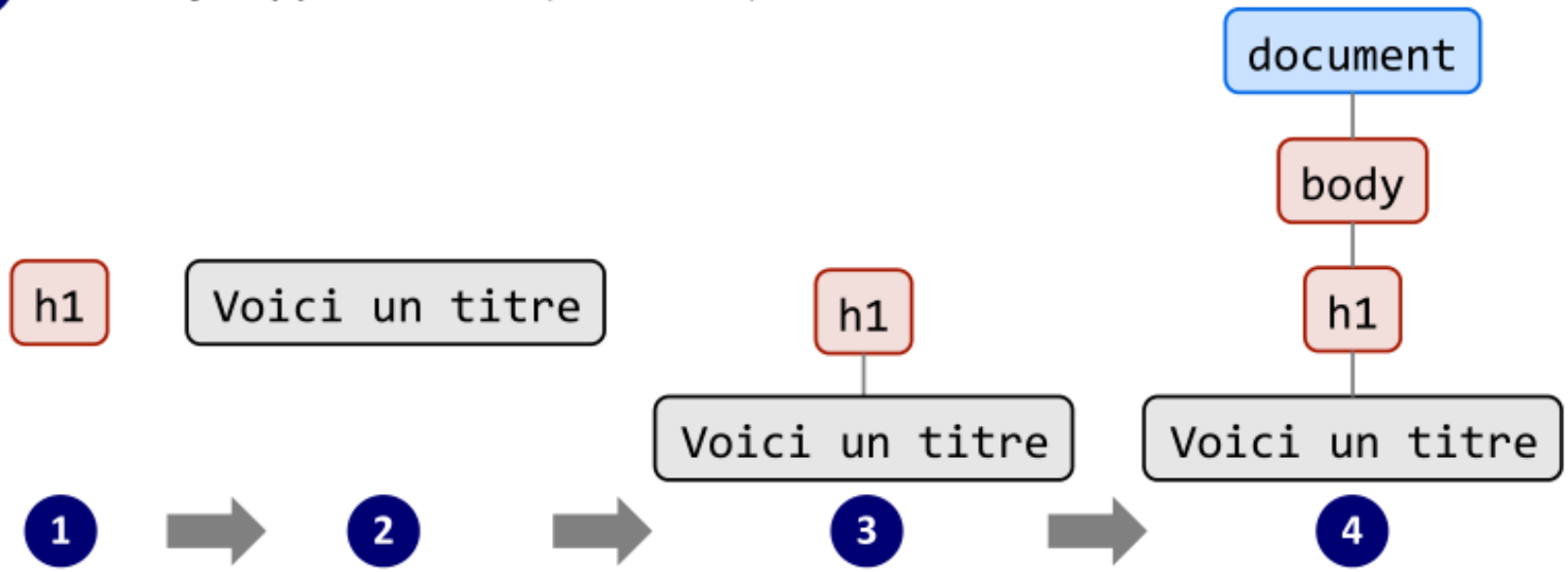
Un peu de texte <a>et un lien <a id=" sdz_link"

href="http://blog.crdp-versailles.fr/rimbaud" title="Découvrez le blog de la Classe Actu

! ">Le Tonnerre de Rimbaud</

Exemple : création d'un nœud <h1>

```
var d = window.document;

1 var noeudH1 = d.createElement("h1");
2 var noeudTexte = d.createTextNode("Voici un titre");

3 noeudH1.appendChild(noeudTexte);

4 d.body.appendChild(noeudH1);
```



ajouter des éléments dans une page

```
// recherche du noeud parent
var divParent = document.getElementById('divParent');

// création des nouveaux noeuds
var nouveauDiv = document.createElement('div');
var nouveauLabel = document.createElement('label');
var nouveauInput = document.createElement('input');

// paramétrage des nouveaux noeuds
nouveauLabel.appendChild(document.createTextNode("Mon nouveau label :"));
nouveauLabel.htmlFor = 'nouveauId';

nouveauInput.name = 'nouveau';
nouveauInput.id = 'nouveauId';
nouveauInput.type = 'text';

// raccord des noeuds
divParent.appendChild(nouveauDiv);
nouveauDiv.appendChild(nouveauLabel);
nouveauDiv.appendChild(nouveauInput);
```

Cloner un élément

- cloneNode() : Cette méthode requiert un paramètre booléen (true ou false) : si vous désirez cloner (copier) le nœud avec (true) ou sans (false) ses enfants et ses différents attributs.
- Exemple : on crée un élément <hr />, et on veut un deuxième, donc on clone le premier :

```
<script>
// On va cloner un élément créé :
var hr1 = document.createElement('hr');
var hr2 = hr1.cloneNode(false); // Il n'a pas d'enfants…

// Ici, on clone un élément existant :
var paragraph1 = document.getElementById('myP');
var paragraph2 = paragraph1.cloneNode(true);

// Et attention, l'élément est cloné, mais pas « inséré »
tant que l'on n'a pas appelé appendChild() :
paragraph1.parentNode.appendChild(paragraph2);
</script>
```

Remplacer un élément par un autre

- replaceChild() : Cette méthode accepte deux paramètres : le 1 est le nouvel élément, et le 2 est l'élément à remplacer. Cette méthode s'utilise sur tous les types de nœuds (éléments, nœuds textuels, etc.).
- Exemple : le contenu textuel (pour rappel, il s'agit du premier enfant de <a>) va être remplacé par « et un hyperlien ». La méthode replaceChild() est exécutée sur l'élément <a>, c'est-à-dire le nœud parent du nœud à remplacer.

```
<body>
  <div>
    <p id="myP">Un peu de texte <a>et un lien</a></p>
  </div>

  <script>
    var link = document.getElementsByTagName('a')[0];
    var newLabel= document.createTextNode('et un hyperlien');

    link.replaceChild(newLabel, link.firstChild);
  </script>
</body>
```

Résultat :

Un peu de texte <a>et un hyperlien

Supprimer un élément

- Pour en supprimer un élément, on utilise *removeChild()*. Cette méthode prend en paramètre le nœud enfant à retirer. Soit le code HTML avec un script ressemble à ceci :

```
1 var link = document.getElementsByTagName('a')[0];
2
3 link.parentNode.removeChild(link);
```

- la méthode removeChild() retourne l'élément supprimé, ce qui veut dire qu'il est parfaitement possible de supprimer un élément HTML pour ensuite le réintégrer où on le souhaite dans le DOM :

```
<script>
var link = document.getElementsByTagName('a')[0];

var oldLink = link.parentNode.removeChild(link);
// On supprime l'élément et on le garde en stock

document.body.appendChild(oldLink);
// On réintègre ensuite l'élément supprimé où on veut et quand on veut
</script>
```

Vérifier la présence d'éléments enfants

- ***hasChildNodes()*** : Il suffit d'utiliser cette méthode sur l'élément de votre choix ; si cet élément possède au moins un enfant, la méthode renverra true :

```
<div>
  <p id="myP">Un peu de texte <a>et un lien</a></p>
</div>

<script>
  var paragraph = document.getElementsByTagName('p')[0];

  alert(paragraph.hasChildNodes()); // Affiche true
</script>
```

insérer un élément avant un autre

- La méthode ***insertBefore()*** permet d'insérer un élément avant un autre. Elle reçoit deux paramètres : le premier est l'élément à insérer, tandis que le deuxième est l'élément avant lequel l'élément va être inséré. Exemple :

```
<p id="myP">Un peu de texte <a>et un lien</a></p>

<script>
  var paragraph    = document.getElementsByTagName('p')[0];
  var emphasis     = document.createElement('em'),
      emphasisText = document.createTextNode(' en emphase légère ');

      emphasis.appendChild(emphasisText);

  paragraph.insertBefore(emphasis, paragraph.lastChild);
</script>
```

Résultat :

Un peu de texte > en emphase légère <a>et un hyperlien

L'objet style

on peut aller plus loin et interagir avec les styles d'une page (en lecture-écriture), en intervenant sur le DOM par des fonctions Javascript agissant sur un nouvel objet : **style** défini sur chaque élément du DOM.

Pour cela il suffit d'obtenir une référence de cet élément : **id**

On peut directement accéder aux attributs id et class d'un élément donné à l'aide des propriétés du DOM **id** et *className.*

L'objet style

il est possible d'avoir accès séparément à toutes les propriétés de style, en faisant appel à la collection style.

L'objet style permet de lire et de modifier de façon dynamique toutes les propriétés de style pour l'élément auquel il s'adresse, avec la syntaxe ***element.style.propriete***.

Il suffit d'en connaitre une référence, par *getElementById()*

Syntaxe La lecture ou la modification des propriétés de style :

nœud= document.getElementById("id");

nœud.style.propriete = "valeur";

Exemple :

en CSS on aurait écrit :

<style type="text/css"> #nœud {font-size:12px; border-bottom: 1px solid #000; }
</style>

l'équivalent en DOM est :

```
var n = document.getElementById("nœud");
// taille de la police de l'élément à 12 pixels.
n.style.fontSize="12px";
// bordure du bas de 1 pixel avec un style solide et une couleur noire.
n.style.borderBottom = "1px solid #000";
```

Remarque : pour chaque propriété dont le nom en CSS comporte un trait d'union, JavaScript demande l'utilisation d'une majuscule. Ainsi on note ***fontSize*** au lieu de font-size

Accès aux classes CSS

Pour connaître, modifier ou bien attribuer l'attribut *CLASS* d'une balise HTML , on utilise la propriété JS **className**.

Exemple:

Zone de Texte dans un div

```
// dans <head>
<style type="text/css">
.out {display:block;width:150px;height:30px;border: 1px solid #000;background-color:none;} .over{display:block;width:150px;height:30px;border: 1px solid #000;background-color:lime;} </style>
// dans <body>
 <div id="divtext" class="out" >Zone de texte dans un div</div>
 <script type="text/javascript">
divtext=document.getElementById("divtext");
// ici, this est exactement divtext
divtext.onmouseover= function() {this.className='over'};
divtext.onmouseout= function() {this.className='out'}
```

La propriété **visibility** qui prend 2 valeurs ***hidden*** ou ***visible.***

l'exemple suivant :

```
<h3 onmouseover="document.getElementById('im2').style.visibility='visible' "
onmouseout="document.getElementById('im2').style.visibility='hidden' ">
Passez la souris ici sur ce paragraphe pour rendre l'image visible ! </h3>
 <p id="im2" style="visibility:hidden" > <img src="style1.png" alt="image"> </p>
```

Si souris ne passe pas sur ce paragraphe

Passez la souris ici sur ce paragraphe pour rendre l'image visible !

Si souris passe sur ce paragraphe